RC4
From Wikipedia, the free encyclopedia

In cryptography, RC4 (also known as ARC4 or ARCFOUR meaning Alleged RC4, see below) is the most widely used software stream cipher and is used in popular protocols such as Secure Sockets Layer (SSL) (to protect Internet traffic) and WEP (to secure wireless networks). While remarkable for its simplicity and speed in software, RC4 has weaknesses that argue against its use in new systems.[2] It is especially vulnerable when the beginning of the output keystream is not discarded, or when nonrandom or related keys are used; some ways of using RC4 can lead to very insecure cryptosystems such as WEP.

Contents

- 1 History
- 2 Description
  - 2.1 The key-scheduling algorithm (KSA)
  - 2.2 The pseudo-random generation algorithm (PRGA)
  - 2.3 Implementation
  - 2.4 Test vectors
- 3 Security
  - 3.1 Roos' biases and key reconstruction from permutation
  - 3.2 Biased outputs of the RC4
  - 3.3 Fluhrer, Mantin and Shamir attack
  - 3.4 Klein's attack
  - 3.5 Combinatorial problem
- 4 RC4 variants
  - 4.1 RC4A
  - 4.2 VMPC
  - 4.3 RC4+
- 5 RC4-based cryptosystems
- 6 See also
- 7 References
- 8 Further reading
- 9 External links

History
RC4 was designed by Ron Rivest of RSA Security in 1987. While it is officially termed "Rivest Cipher 4", the RC acronym is alternatively understood to stand for "Ron's Code"[3] (see also RC2, RC5 and RC6).

RC4 was initially a trade secret, but in September 1994 a description of it was anonymously posted to the Cypherpunks mailing list.[4] It was soon posted on the sci.crypt newsgroup, and from there to many sites on the Internet. The leaked code was confirmed to be genuine as its output was found to match that of proprietary software using licensed RC4. Because the algorithm is known, it is no longer a trade secret. The name RC4 is trademarked, so RC4 is often referred to as ARCFOUR or ARC4 (meaning alleged RC4) to avoid trademark problems. RSA Security has never officially released the algorithm; Rivest has, however, linked to the English Wikipedia article on RC4 in his own course notes.[5] RC4 has become part of some commonly used encryption protocols and standards, including WEP and WPA for wireless cards and TLS.

The main factors in RC4's success over such a wide range of applications are its speed and simplicity: efficient implementations in both software and hardware are very easy to develop.

**Description**

RC4 generates a pseudorandom stream of bits (a keystream). As with any stream cipher, these can be used for encryption by combining it with the plaintext using bit-wise exclusive-or; decryption is performed the same way (since exclusive-or with given data is an involution). (This is similar to the Vernam cipher except that generated pseudorandom bits, rather than a prepared stream, are used.) To generate the keystream, the cipher makes use of a secret internal state which consists of two parts:

1. A permutation of all 256 possible bytes (denoted "S" below).
2. Two 8-bit index-pointers (denoted "i" and "j").

The permutation is initialized with a variable length key, typically between 40 and 256 bits, using the key-scheduling algorithm (KSA). Once this has been completed, the stream of bits is generated using the pseudo-random generation algorithm (PRGA).

**The key-scheduling algorithm (KSA)**

The key-scheduling algorithm is used to initialize the permutation in the array "S". "keylength" is defined as the number of bytes in the key and can be in the range 1 ≤ keylength ≤ 256, typically between 5 and 16, corresponding to a key length of 40 – 128 bits. First, the array "S" is initialized to the identity permutation. S is then processed for 256 iterations in a similar way to the main PRGA, but also mixes in bytes of the key at the same time.

```plaintext
for i from 0 to 255
    S[i] := i
endfor

j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor
```

**The pseudo-random generation algorithm (PRGA)**
For as many iterations as are needed, the PRGA modifies the state and outputs a byte of the keystream. In each iteration, the PRGA increments \( i \), looks up the \( i \)th element of \( S \), \( S[i] \), and adds that to \( j \), exchanges the values of \( S[i] \) and \( S[j] \), and then uses the sum \( S[i] + S[j] \) (modulo 256) as an index to fetch a third element of \( S \), (the keystream value \( K \) below) which is XORed with the next byte of the message to produce the next byte of either ciphertext or plaintext. Each element of \( S \) is swapped with another element at least once every 256 iterations.

```plaintext
\[
i := 0
\]
\[
j := 0
\]
\[
\text{while GeneratingOutput:
  \quad i := (i + 1) \mod 256
  \quad j := (j + S[i]) \mod 256
  \quad \text{swap values of } S[i] \text{ and } S[j]
  \quad k := \text{inputByte XOR } S[(S[i] + S[j]) \mod 256]
  \quad \text{output } K
\]
\]
```

**Implementation**

Many stream ciphers are based on linear feedback shift registers (LFSRs), which, while efficient in hardware, are less so in software. The design of RC4 avoids the use of LFSRs, and is ideal for software implementation, as it requires only byte manipulations. It uses 256 bytes of memory for the state array, \( S[0] \) through \( S[255] \), \( k \) bytes of memory for the key, key[0] through key[k-1], and integer variables, \( i \), \( j \), and \( K \). Performing a modular reduction of some value modulo 256 can be done with a bitwise AND with 255 (which is equivalent to taking the low-order byte of the value in question).

**Test vectors**

These test vectors are not official, but convenient for anyone testing their own RC4 program. The keys and plaintext are ASCII, the keystream and ciphertext are in hexadecimal.

<table>
<thead>
<tr>
<th>Key</th>
<th>Keystream</th>
<th>Plaintext</th>
<th>Ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wiki</td>
<td>6044DB6D41B7...</td>
<td>pedia</td>
<td>1021BF0420</td>
</tr>
<tr>
<td>Secret</td>
<td>04D46B053CA87B59...</td>
<td>Attack at dawn</td>
<td>45A01F645FC35B383552544B9BF5</td>
</tr>
<tr>
<td>Key</td>
<td>EB9F7781B734CA72A719...</td>
<td>Plaintext</td>
<td>BBF316E8D940AF0AD3</td>
</tr>
</tbody>
</table>

[RC4 - Wikipedia, the free encyclopedia](http://en.wikipedia.org/wiki/RC4)
Security

Unlike a modern stream cipher (such as those in eSTREAM), RC4 does not take a separate nonce alongside the key. This means that if a single long-term key is to be used to securely encrypt multiple streams, the cryptosystem must specify how to combine the nonce and the long-term key to generate the stream key for RC4. One approach to addressing this is to generate a "fresh" RC4 key by hashing a long-term key with a nonce. However, many applications that use RC4 simply concatenate key and nonce; RC4's weak key schedule then gives rise to a variety of serious problems.

Because RC4 is a stream cipher, it is more malleable than common block ciphers. If not used together with a strong message authentication code (MAC), then encryption is vulnerable to a bit-flipping attack. It is noteworthy, however, that RC4, being a stream cipher, is the only common cipher which is immune[6] to the 2011 BEAST attack on TLS 1.0, which exploits a known weakness in the way cipher block chaining mode is used with all of the other ciphers supported by TLS 1.0, which are all block ciphers.

Roos' biases and key reconstruction from permutation

In 1995, Andrew Roos experimentally observed that the first byte of the keystream is correlated to the first three bytes of the key and the first few bytes of the permutation after the KSA are correlated to some linear combination of the key bytes.[7] These biases remained unproven until 2007, when Goutam Paul, Siddheshwar Rathi and Subhamoy Maitra[8] proved the keystream-key correlation and in another work Goutam Paul and Subhamoy Maitra[9] proved the permutation-key correlations. The latter work also used the permutation-key correlations to design the first algorithm for complete key reconstruction from the final permutation after the KSA, without any assumption on the key or IV. This algorithm has a constant probability of success in a time which is the square root of the exhaustive key search complexity. Subsequently, many other works have been performed on key reconstruction from RC4 internal states.[10][11][12] Subhamoy Maitra and Goutam Paul[13] also showed that the Roos type biases still persist even when one considers nested permutation indices, like $S[S[i]]$ or $S[S[S[i]]]$. These types of biases are used in some of the later key reconstruction methods for increasing the success probability.

Biased outputs of the RC4

The keystream generated by the RC4 is biased in varying degrees towards certain sequences. The best such attack is due to Itsik Mantin and Adi Shamir who showed that the second output byte of the cipher was biased toward zero with probability 1/128 (instead of 1/256). This is due to the fact that if the third byte of the original state is zero, and the second byte is not equal to 2, then the second output byte is always zero. Such bias can be detected by observing only 256 bytes.[14]

Souradyuti Paul and Bart Preneel of COSIC showed that the first and the second bytes of the RC4 were also biased. The number of required samples to detect this bias is $2^{25}$ bytes.[15]

Scott Fluhrer and David McGrew also showed such attacks which distinguished the keystream of the RC4 from a random stream given a gigabyte of output.[16]

The complete characterization of a single step of RC4 PRGA was performed by Riddhipratim Basu, Shirshendu Ganguly, Subhamoy Maitra, and Goutam Paul.[17] Considering all the permutations, they prove...
that the distribution of the output is not uniform given i and j, and as a consequence, information about j is always leaked into the output.

**Fluhrer, Mantin and Shamir attack**

*Main article: Fluhrer, Mantin and Shamir attack*

In 2001, a new and surprising discovery was made by Fluhrer, Mantin and Shamir: over all possible RC4 keys, the statistics for the first few bytes of output keystream are strongly non-random, leaking information about the key. If the long-term key and nonce are simply concatenated to generate the RC4 key, this long-term key can be discovered by analysing a large number of messages encrypted with this key.[18] This and related effects were then used to break the WEP ("wired equivalent privacy") encryption used with 802.11 wireless networks. This caused a scramble for a standards-based replacement for WEP in the 802.11 market, and led to the IEEE 802.11i effort and WPA.[19]

Cryptosystems can defend against this attack by discarding the initial portion of the keystream. Such a modified algorithm is traditionally called "RC4-drop[n]", where n is the number of initial keystream bytes that are dropped. The SCAN default is n = 768 bytes, but a conservative value would be n = 3072 bytes.[20]

The Fluhrer, Mantin and Shamir attack does not apply to RC4-based SSL, since SSL generates the encryption keys it uses for RC4 by hashing, meaning that different SSL sessions have unrelated keys.[21]

**Klein's attack**

In 2005, Andreas Klein presented an analysis of the RC4 stream cipher showing more correlations between the RC4 keystream and the key.[22] Erik Tews, Ralf-Philipp Weinmann, and Andrei Pychkine used this analysis to create aircrack-ptw, a tool which cracks 104-bit RC4 used in 128-bit WEP in under a minute.[23] Whereas the Fluhrer, Mantin, and Shamir attack used around 10 million messages, aircrack-ptw can break 104-bit keys in 40,000 frames with 50% probability, or in 85,000 frames with 95% probability.

**Combinatorial problem**

A combinatorial problem related to the number of inputs and outputs of the RC4 cipher was first posed by Itsik Mantin and Adi Shamir in 2001, whereby, of the total 256 elements in the typical state of RC4, if \( x \) number of elements (\( x \leq 256 \)) are only known (all other elements can be assumed empty), then the maximum number of elements that can be produced deterministically is also \( x \) in the next 256 rounds. This conjecture was put to rest in 2004 with a formal proof given by Souradyuti Paul and Bart Preneel.[24]

**RC4 variants**

As mentioned above, the most important weakness of RC4 comes from the insufficient key schedule; the first bytes of output reveal information about the key. This can be corrected by simply discarding some initial portion of the output stream.[25] This is known as RC4-drop\( N \), where \( N \) is typically a multiple of 256, such as 768 or 1024.

A number of attempts have been made to strengthen RC4, notably RC4A, VMPC, and RC4+.
RC4A

Souradyuti Paul and Bart Preneel have proposed an RC4 variant, which they call RC4A.\textsuperscript{[26]}

RC4A uses two state arrays $S_1$ and $S_2$, and two indexes $j_1$ and $j_2$. Each time $i$ is incremented, two bytes are generated:

1. First, the basic RC4 algorithm is performed using $S_1$ and $j_1$, but in the last step, $S_1[i] + S_1[j_1]$ is looked up in $S_2$.
2. Second, the operation is repeated (without incrementing $i$ again) on $S_2$ and $j_2$, and $S_1[S_2[i]+S_2j_2]$ is output.

Thus, the algorithm is:

\begin{verbatim}
all arithmetic is performed modulo 256
i := 0
j1 := 0
j2 := 0
while GeneratingOutput:
  i := i + 1
  j1 := j1 + S1[i]
  swap values of S1[i] and S1[j1]
  output S2[S1[i] + S1[j1]]
  j2 := j2 + S2[i]
  swap values of S2[i] and S2[j2]
  output S1[S2[i] + S2[j2]]
endwhile
\end{verbatim}

Although the algorithm required the same number of operations per output byte, there is greater parallelism than RC4, providing a possible speed improvement.

Although stronger than RC4, this algorithm has also been attacked,\textsuperscript{[27]} with Alexander Maximov\textsuperscript{[28]} and a team from NEC\textsuperscript{[29]} developing ways to distinguish its output from a truly random sequence.

VMPC

"Variably Modified Permutation Composition" is another RC4 variant.\textsuperscript{[30]} It uses the same key schedule as RC4, but iterating 768 times rather than 256 (it is not the same as RC4-drop512 because all iterations incorporate key material), and with an optional additional 768 iterations to incorporate an initial vector. Written to highlight the similarity to RC4 as much as possible, the output generation function operates as follows:

\begin{verbatim}
All arithmetic is performed modulo 256.
i := 0
while GeneratingOutput:
  a := S[i]
  j := S[j + a]
  b := S[j]
  output S[S[b + 1]]
  S[i] := b     (Swap S[i] and S[j])
  S[j] := a
  i := i + 1
endwhile
\end{verbatim}

This was attacked in the same papers as RC4A.\textsuperscript{[27][29]}
RC4+

RC4+ is a modified version of RC4 with a more complex three-phase key schedule (taking about $3 \times$ as long as RC4, or the same as RC4-drop512), and a more complex output function which performs four additional lookups in the S array for each byte output, taking approximately $1.7 \times$ as long as basic RC4.[31]

```
All arithmetic modulo 256. << and >> are left and right shift, ⊕ is exclusive OR

while GeneratingOutput:
    i := i + 1
    a := S[i]
    j := j + a
    b := S[j]
    S[i] := b  (Swap S[i] and S[j])
    S[j] := a
    c := S[i<<5 ⊕ j>>3] + S[j<<5 ⊕ i>>3]
    output (S[a+b] + S[c⊕0xAA]) ⊕ S[j+b]
endwhile
```

This algorithm has not been analyzed significantly.

RC4-based cryptosystems

- WEP
- WPA (default algorithm, but can be configured to use AES-CCMP instead of RC4)
- BitTorrent protocol encryption
- Microsoft Point-to-Point Encryption
- Opera Mini[32]
- Secure Sockets Layer (optionally)
- Secure shell (optionally)
- Remote Desktop Protocol
- Kerberos (optionally)
- SASL Mechanism Digest-MD5 (optionally)
- Gpcode.AK, an early June 2008 computer virus for Microsoft Windows, which takes documents hostage for ransom by obscuring them with RC4 and RSA-1024 encryption
- PDF
- Skype (in modified form)[33]

Where a cryptosystem is marked with "(optionally)", RC4 is one of several ciphers the system can be configured to use.

See also

- eSTREAM - An evaluation of new stream ciphers being conducted by the EU.
- TEA, Block TEA also known as eXtended TEA and Corrected Block TEA - A family of block ciphers that, like RC4, are designed to be very simple to implement.
- Advanced Encryption Standard
- CipherSaber

References


7. ^ Andrew Roos. A Class of Weak Keys in the RC4 Stream Cipher. Two posts in sci.crypt, message-id 43u1eh$1j3@hermes.is.co.za and 44ebge$llf@hermes.is.co.za, 1995.


19. ^ Interim technology for wireless LAN security: WPA to replace WEP while industry develops new security standard (http://findarticles.com/p/articles/mi_m0DIS/is_2003_Jan/ai_n27590035/)

20. ^ "RC4-drop(nbytes)" (http://www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html#RC4-drop) in the "Standard Cryptographic Algorithm Naming" database


27. ^ a b CryptoLounge: RC4A (http://www.cryptolounge.org/wiki/RC4A)


29. ^ a b Yukiyasu Tsunoo; Teruo Saito; Hiroyasu Kubo; Maki Shigeru; Tomoyasu Suzuki; Takeshi Kawabata (2005), The Most Efficient Distinguishing Attack on VMPC and RC4A (http://www.ecrypt.eu.org/stream/papersdir/037.pdf), http://www.ecrypt.eu.org/stream/papersdir/037.pdf


Further reading


External links

- RC4 Implementation in Delphi (http://www.delphibasics.info/home/delphibasicssnippets/delphirc4sourcecodebysteve10120)
- SCAN's entry for RC4 (http://www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html#RC4)
- RC4 - Cryptology Pointers by Helger Lipmaa (http://research.cyber.ee/~lipmaa/crypto/link/stream/rc4.php)

RC4 in WEP
- Fluhrer, Mantin, and Shamir attack on WEP (postscript format) (http://www.wisdom.weizmann.ac.il/~itsik/RC4/Papers/rc4_wep.ps)


Categories: Stream ciphers | Broken stream ciphers | Pseudorandom number generators | Free ciphers

- This page was last modified on 14 November 2012 at 16:38.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. See Terms of Use for details. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.